University of Technology
الجامعة التكنولوجية

Computer Science Department
قسم علوم الحاسوب

Multimedia File Format - Lab
بنية ملف الوسائط المتعددة

Lect. Teaba wala aldeen khairi
م. طيبة ولاء الدين خيري

cs.uotechnology.edu.iq

Lecture 1 Text File Format

DOC File Format | .doc Extension

The doc file format, which has been in play for decades now, is a common text-based document format recognized around the world. This article will look at what a .doc file is, how it can be used, opening and creating .doc files, its advantages, and applications that use this format.

What is the .doc file format?

A proprietary text document called .doc file format was created and used by Microsoft. The program appeared in the early 1990s within the Microsoft Word software. “DOC” means “document,” which is usually related to Microsoft Word, and is flexible enough for word processing, it can be text or formats among others.

Uses of the .doc File Format

.doc is a file that stores textual documents. Some common uses include:

- Word Processing: Word documents in the format of .doc are widely used for building, revising, and adjusting textual documents – letters, reports, essays, articles, etc.
- Documentation: Organizations such as policies, and procedures for the user manuals of many of them, use .doc files.
- Academic Work: Researchers and students mostly use .doc files for writing f research papers, theses, and essays.

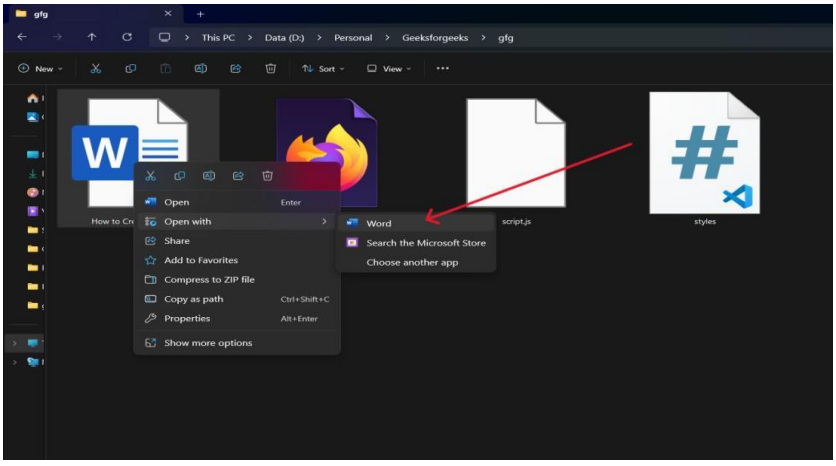
- Business Correspondence: Business letters, memos, and other written communication are typically conducted using the .doc format.
- Creative Writing: Novels, short stories, and poems are drafted and edited on .doc files by authors and writers.
- Legal Documents: There are legal documents, contracts, and agreements that lawyers and legal professionals design in .doc format.

How to Open .doc Files

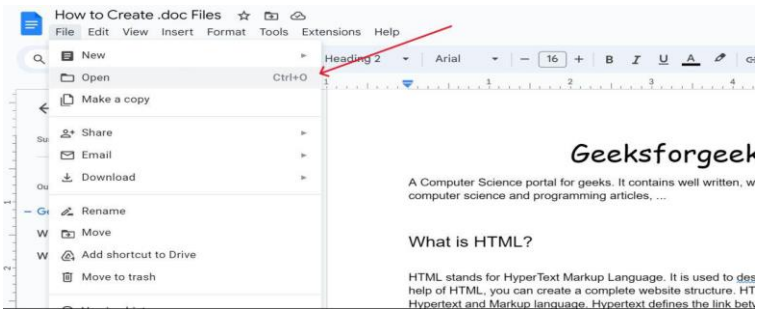
Opening .doc files is straightforward, and you have several options:

- Microsoft Word: The most frequently used application for opening .doc file is Microsoft Word, which forms an integral part of the Microsoft Office suite. Double-click on the .doc file if there is Word installed, and your document will open in Word.
- Microsoft Word Online: Opening .doc files works with the free online version of Microsoft Word. For easy reading, you can just upload the file into your OneDrive and open it with Word Online.
- LibreOffice: The open source office suite known as, LibreOFFice can open and edit .doc files. There is a free alternative to MS office.
- Google Docs: Google Docs is a web word processor that open and edits .doc file. It is possible to upload the file to Google Drive, then open it in Google Docs.

To Open double type on file or right click on mouse then select open with choose word or any other software then click on word then it will open the file



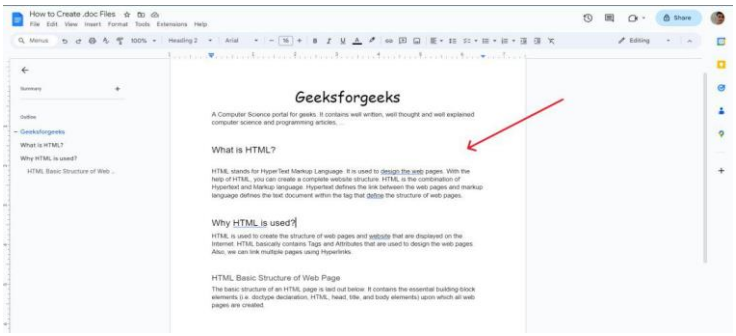
To Open same file on google docs first visit https://docs.google.com/ then login after that on left side click on FILE after that click on Open or just simply use shortcut Ctrl + O



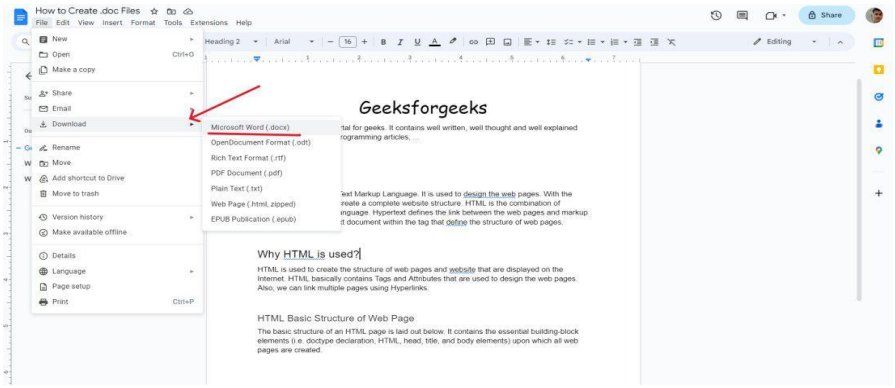
How to Create .doc Files

Open your Microsoft word or google docs processing software.

- Create a new document or open an existing one.
- Type or paste your content into the document.
- Format the text, add images, and customize the layout as needed.



Save the document and choose the .doc file format as the file type when saving. Then click on Microsoft Word (.docx)



Benefits of the .doc File Format

The .doc file format offers several benefits:

- Compatibility: .doc files are also very popular because most existing word processors can support them, and this is helpful whenever one needs collaboration on a certain document.
- Rich Formatting: We can easily write eye-catching documents using fonts, styles, images, tables etc.
- Editing Capabilities: This is because the .doc files are easy to edit, thus letting you change the document if need be.
- Security: It also allows you to add password-protection to a .doc file to secure private data.

Open text file format using matlab

1- Open

```
fileID = fopen('filename.txt', 'r'); % 'r' indicates read mode
if fileID == -1
    error('File cannot be opened. Check the file path.');
end
```

2- Reading

```
while ~feof(fileID) % Loop until the end of the file
    line = fgets(fileID); % Read one line
    disp(line); % Display the line
end
```

3- Closing

```
fclose(fileID);
```

Example of reading and printing a file

```
% Open the file
fileID = fopen('example.txt', 'r');
if fileID == -1
   error('File cannot be opened. Check the file path.');
end

% Read and print the file line by line
while ~feof(fileID)
   line = fgets(fileID); % Read one line
   disp(line); % Display the line
end

% Close the file
fclose(fileID);
```

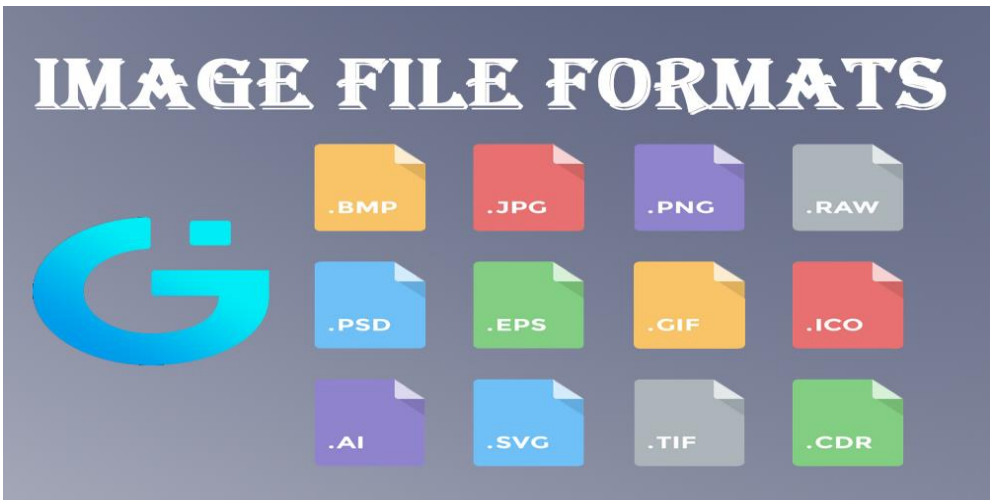
Lab Assignment:

1- Create a Google docs online and write a simplr CV about yourself .
2- Create txt dox contain a method about Green computing .
3- Write a program to print a text contain information about computer science department .

Lecture 2 Image File Format

Image Formats

Image Format describes how data related to the image will be stored. Data can be stored in compressed, Uncompressed, or vector format. Each format of the image has a different advantage and disadvantage. Image types such as TIFF are good for printing while JPG or PNG, are best for the web.



1- Loade image

```
image = imread('image_file.jpg'); % Replace 'image_file.jpg' with
your image file name
```

2- Display

```
imshow(image);
```

3- Image information

```
info = size(image); % Returns the dimensions of the image
disp(['Image Dimensions: ', num2str(info(1)), ' x ',
num2str(info(2))]);
```

4- Example

```
% Load the image
image = imread('example.jpg'); % Replace 'example.jpg'
with your image file

% Display the image
imshow(image);
title('Loaded Image'); % Add a title to the figure

% Get image information
info = size(image);
disp(['Image Dimensions: ', num2str(info(1)), ' x ',
num2str(info(2)), ' pixels']);
```

1. Reading a BMP File

```
% Load a BMP file
bmpImage = imread('example.bmp'); % Replace 'example.bmp' with your
BMP file path
```

2. Displaying a BMP File

```
imshow(bmpImage);
title('BMP Image');
```

3. Writing a BMP File

```
% Save the image as a BMP file
imwrite(bmpImage, 'output.bmp');
disp('Image saved as output.bmp');
```

4. Checking File Properties

```
info = imfinfo('example.bmp');
disp(info); % Display all metadata
```

Example output

```
Filename: 'example.bmp'
FileSize: 123456 bytes
Width: 800
Height: 600
BitDepth: 24
ColorType: 'truecolor'
```

5- Grayscale BMP Files

```
% Convert the color BMP to grayscale
grayImage = rgb2gray(bmpImage);
imshow(grayImage);
title('Grayscale BMP Image');
```

6- Full Workflow

```
7- % Step 1: Read a BMP file
bmpImage = imread('example.bmp');

% Step 2: Display the BMP file
imshow(bmpImage);
title('Original BMP Image');

% Step 3: Get BMP file information
info = imfinfo('example.bmp');
disp('File Information:');
disp(info);

% Step 4: Convert to grayscale (if it's a color image)
if size(bmpImage, 3) == 3 % Check if the image has 3 color channels
   grayImage = rgb2gray(bmpImage);
   figure; % Open a new figure window
   imshow(grayImage);
   title('Grayscale BMP Image');
end
% Step 5: Save the grayscale image as a new BMP file
imwrite(grayImage, 'grayscale_output.bmp');
disp('Grayscale BMP image saved as grayscale_output.bmp');
```

BMP File Header

Notes

1. **Endianess:** BMP files use **little-endian** byte order. MATLAB's typecast function handles this automatically.
2. **File Size Validation:** Ensure the fileSize matches the actual file size.
3. **Header Size Variations:** This example assumes the BMP uses the common 40-byte BITMAPINFOHEADER. Other BMP versions may have different header sizes.

```
% Open the BMP file in binary mode
fileID = fopen('example.bmp', 'r');
if fileID == -1
    error('Cannot open the BMP file. Check the file path.');
end

% Read the File Header (14 bytes)
fileHeader = fread(fileID, 14, 'uint8');

% Read the DIB Header (40 bytes for BITMAPINFOHEADER)
dibHeader = fread(fileID, 40, 'uint8');

% Close the file
fclose(fileID);

% Parse the File Header
signature = char(fileHeader(1:2)'); % Should be 'BM'
fileSize = typecast(fileHeader(3:6), 'uint32'); % File size in bytes
dataOffset = typecast(fileHeader(11:14), 'uint32'); % Offset to image data

% Parse the DIB Header
headerSize = typecast(dibHeader(1:4), 'uint32'); % DIB header size
imageWidth = typecast(dibHeader(5:8), 'int32'); % Image width in pixels
imageHeight = typecast(dibHeader(9:12), 'int32'); % Image height in pixels
bitDepth = typecast(dibHeader(15:16), 'uint16'); % Bits per pixel

% Display the parsed header information
disp('BMP Header Information:');
```

```
fprintf('Signature: %s\n', signature);
fprintf('File Size: %d bytes\n', fileSize);
fprintf('Data Offset: %d bytes\n', dataOffset);
fprintf('DIB Header Size: %d bytes\n', headerSize);
fprintf('Image Width: %d pixels\n', imageWidth);
fprintf('Image Height: %d pixels\n', imageHeight);
fprintf('Bit Depth: %d bits per pixel\n', bitDepth);
```

The output will be:

BMP Header Information:

Signature: BM

File Size: 786486 bytes

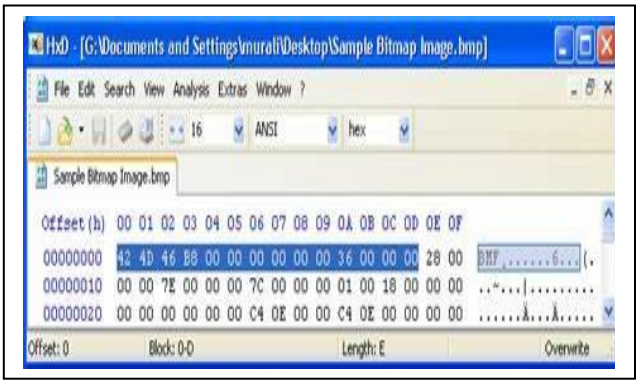
Data Offset: 54 bytes

DIB Header Size: 40 bytes

Image Width: 800 pixels

Image Height: 600 pixels

Bit Depth: 24 bits per pixel



Notes on BMP in MATLAB

- BMP files are uncompressed by default, making them large but preserving image quality.
- MATLAB supports both RGB and grayscale BMP images.
- For color BMP images, the data is stored as a 3D matrix (MxNx3), while for grayscale, it's a 2D matrix (MxN).
- Let me know if you'd like to explore any specific aspect of BMP file handling in MATLAB! 😊

Lecture 3 Audio File format

Audio format defines the quality and loss of audio data. Based on application different type of audio format are used. Audio formats are broadly divided into three parts:

1- Uncompressed Format
2- Lossy Compressed format
3- Lossless Compressed Format

1- Reading an Audio File

```
% Read an audio file
[audioData, sampleRate] = audioread('example.wav'); % Replace with your file name

% Display basic information
disp(['Sample Rate: ', num2str(sampleRate), ' Hz']);
disp(['Audio Data Size: ', num2str(size(audioData, 1)), ' samples']);
```

2- Playing an Audio File

```
% Play the audio file
sound(audioData, sampleRate);

% Alternatively, using audioplayer for more control
player = audioplayer(audioData, sampleRate);
play(player);
```

3- Writing an Audio File

```
% Save audio data as a new file
audiowrite('output.wav', audioData, sampleRate);
disp('Audio file saved as output.wav');
```

4- Visualizing Audio Data

```
% Plot the audio waveform
time = (0:length(audioData)-1) / sampleRate; % Time vector
plot(time, audioData);
xlabel('Time (s)');
ylabel('Amplitude');
title('Audio Waveform');
```

5- Reading File Metadata

```
info = audioinfo('example.wav'); % Replace with your file name
disp(info);
```

6- Converting Between File Formats

```
% Read an MP3 file
[audioData, sampleRate] = audioread('example.mp3');

% Save as a WAV file
audiowrite('converted.wav', audioData, sampleRate);
disp('MP3 file converted to WAV format');
```

Audio Signal Processing Examples

a- Changing Playback Speed

```
% Play at half the speed
sound(audioData, sampleRate / 2);
```

b- Normalizing Audio

```
audioData = audioData / max(abs(audioData));
```

c- Extracting a Segment

```
startSample = round(2 * sampleRate); % Start at 2 seconds
endSample = round(4 * sampleRate);   % End at 4 seconds
segment = audioData(startSample:endSample);

% Play the segment
sound(segment, sampleRate);
```

Complete Example Code

```
% Step 1: Read an audio file
[audioData, sampleRate] = audioread('example.wav');

% Step 2: Display basic information
disp(['Sample Rate: ', num2str(sampleRate), ' Hz']);
disp(['Duration: ', num2str(length(audioData) / sampleRate), ' seconds']);

% Step 3: Plot the waveform
```

```
time = (0:length(audioData)-1) / sampleRate;
plot(time, audioData);
xlabel('Time (s)');
ylabel('Amplitude');
title('Audio Waveform');

% Step 4: Play the audio
sound(audioData, sampleRate);

% Step 5: Save a normalized version as a new file
audioData = audioData / max(abs(audioData));
audiowrite('normalized_output.wav', audioData, sampleRate);
disp('Normalized audio saved as normalized_output.wav');
```

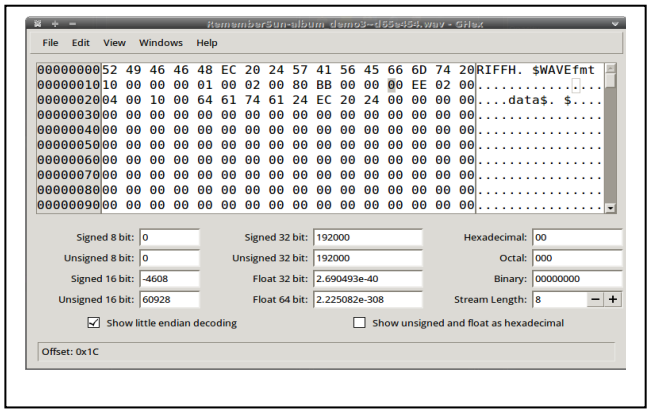
Code to Read and Parse the WAV Header in MATLAB

```
% Open the WAV file in binary mode
fileID = fopen('example.wav', 'r'); % Replace 'example.wav' with your file name
if fileID == -1
    error('Cannot open WAV file. Check the file path.');
end

% Read the RIFF Header (12 bytes)
riffHeader = fread(fileID, 12, 'uint8');

% Parse RIFF Header
chunkID = char(riffHeader(1:4)'); % Should be 'RIFF'
chunkSize = typecast(riffHeader(5:8), 'uint32'); % File size - 8 bytes
format = char(riffHeader(9:12)'); % Should be 'WAVE'

% Read the FMT Subchunk (24 bytes or more)
fmtHeader = fread(fileID, 24, 'uint8');

% Parse FMT Subchunk
subchunk1ID = char(fmtHeader(1:4)'); % Should be 'fmt '
subchunk1Size = typecast(fmtHeader(5:8), 'uint32'); % Size of the fmt chunk
audioFormat = typecast(fmtHeader(9:10), 'uint16'); % Audio format (1 = PCM)
numChannels = typecast(fmtHeader(11:12), 'uint16'); % Number of channels
sampleRate = typecast(fmtHeader(13:16), 'uint32'); % Sample rate
byteRate = typecast(fmtHeader(17:20), 'uint32'); % Byte rate
blockAlign = typecast(fmtHeader(21:22), 'uint16'); % Block align
bitsPerSample = typecast(fmtHeader(23:24), 'uint16'); % Bits per sample

% Read the Data Subchunk header
dataHeader = fread(fileID, 8, 'uint8');
```

```
% Parse Data Subchunk
subchunk2ID = char(dataHeader(1:4)'); % Should be 'data'
subchunk2Size = typecast(dataHeader(5:8), 'uint32'); % Data size in bytes

% Close the file
fclose(fileID);

% Display the parsed header information
disp('WAV Header Information:');
fprintf('Chunk ID: %s\n', chunkID);
fprintf('Chunk Size: %d bytes\n', chunkSize);
fprintf('Format: %s\n', format);
fprintf('Subchunk1 ID: %s\n', subchunk1ID);
fprintf('Subchunk1 Size: %d bytes\n', subchunk1Size);
fprintf('Audio Format: %d (1 = PCM)\n', audioFormat);
fprintf('Number of Channels: %d\n', numChannels);
fprintf('Sample Rate: %d Hz\n', sampleRate);
fprintf('Byte Rate: %d bytes/second\n', byteRate);
fprintf('Block Align: %d bytes\n', blockAlign);
fprintf('Bits Per Sample: %d bits\n', bitsPerSample);
fprintf('Subchunk2 ID: %s\n', subchunk2ID);
fprintf('Subchunk2 Size: %d bytes\n', subchunk2Size);
```

The output will be

```
WAV Header Information:
Chunk ID: RIFF
Chunk Size: 882036 bytes
Format: WAVE
Subchunk1 ID: fmt
Subchunk1 Size: 16 bytes
Audio Format: 1 (1 = PCM)
Number of Channels: 2
Sample Rate: 44100 Hz
Byte Rate: 176400 bytes/second
Block Align: 4 bytes
Bits Per Sample: 16 bits
Subchunk2 ID: data
Subchunk2 Size: 882000 bytes
```



Lecture 4 Video File format

MATLAB provides tools for reading, processing, and writing video files using functions like VideoReader and VideoWriter. These tools support common video formats such as **MP4**, **AVI**, and **MOV**, depending on the installed codecs and your MATLAB version.

Working with Video Files in MATLAB

1. Reading Video Files

Use the VideoReader class to load a video file and read its frames.

```
% Read a video file
videoFile = 'example.mp4'; % Replace with your video file name
videoReader = VideoReader(videoFile);

% Display video properties
disp('Video Properties:');
disp(['Filename: ', videoReader.Name]);
disp(['Duration: ', num2str(videoReader.Duration), ' seconds']);
disp(['Frame Rate: ', num2str(videoReader.FrameRate), ' fps']);
disp(['Width: ', num2str(videoReader.Width), ' pixels']);
disp(['Height: ', num2str(videoReader.Height), ' pixels']);
```

2. Reading and Displaying Frames

```
% Loop through each frame
while hasFrame(videoReader)
    frame = readFrame(videoReader); % Read a frame
    imshow(frame); % Display the frame
    pause(1 / videoReader.FrameRate); % Pause to match frame rate
end
```

4. Writing a Video File

```
% Create a VideoWriter object
outputVideo = VideoWriter('output.mp4', 'MPEG-4'); % Specify
output format
outputVideo.FrameRate = 30; % Set frame rate
open(outputVideo);

% Write frames to the video
for k = 1:100
   % Example: Create a simple frame (black-to-white gradient)
   frame = uint8(255 * (k / 100) * ones(240, 320, 3));
   writeVideo(outputVideo, frame); % Write frame to video
end

close(outputVideo);
disp('Video written to output.mp4');
```

5. Extracting Specific Frames

```
% Get a specific frame by time
timestamp = 2.5; % Time in seconds
videoReader.CurrentTime = timestamp;
frame = readFrame(videoReader);

% Display the frame
imshow(frame);
title(['Frame at ', num2str(timestamp), ' seconds']);
```

Complete Example: Reading, Processing, and Writing a Video

```
% Read the video file
videoReader = VideoReader('example.mp4');
outputVideo = VideoWriter('processed_video.mp4', 'MPEG-4');
outputVideo.FrameRate = videoReader.FrameRate;
open(outputVideo);

% Process each frame
while hasFrame(videoReader)
    frame = readFrame(videoReader); % Read a frame

    % Example: Convert the frame to grayscale
    grayFrame = rgb2gray(frame);

    % Convert back to RGB to write to the video (VideoWriter requires RGB
frames)
    processedFrame = repmat(grayFrame, 1, 1, 3);
    writeVideo(outputVideo, processedFrame); % Write the processed frame
end

% Close the video writer
close(outputVideo);
disp('Processed video written to processed_video.mp4');
```

Key Notes

1. **Dependencies**: Ensure you have the required codecs installed on your system for reading and writing specific video formats.
2. **Performance**: For long videos, consider processing only a subset of frames to save time and memory.
3. **Video Analysis**: MATLAB also supports advanced video analysis tasks such as object detection, tracking, and motion analysis using the **Computer Vision Toolbox**.

4. Let me know if you'd like to explore specific video-processing tasks in MATLAB! 😊

AVI File Header Structure

An AVI file has the following main components in its header:

```
% Open the AVI file in binary mode
fileID = fopen('example.avi', 'r'); % Replace 'example.avi' with your file path
if fileID == -1
    error('Cannot open AVI file. Check the file path.');
end

% Read the RIFF Header
riffHeader = fread(fileID, 12, 'uint8');

% Parse RIFF Header
chunkID = char(riffHeader(1:4)'); % Should be 'RIFF'
chunkSize = typecast(riffHeader(5:8), 'uint32'); % File size - 8 bytes
format = char(riffHeader(9:12)'); % Should be 'AVI '

% Read the LIST Header
listHeader = fread(fileID, 12, 'uint8');

% Parse LIST Header
listID = char(listHeader(1:4)'); % Should be 'LIST'
listSize = typecast(listHeader(5:8), 'uint32');
listType = char(listHeader(9:12)'); % Should be 'hdrl'

% Read the AVIMAINHEADER (56 bytes)
aviMainHeader = fread(fileID, 56, 'uint8');

% Parse AVIMAINHEADER
microSecPerFrame = typecast(aviMainHeader(1:4), 'uint32'); %
Microseconds per frame
maxBytesPerSec = typecast(aviMainHeader(5:8), 'uint32'); % Max bytes per
second
```

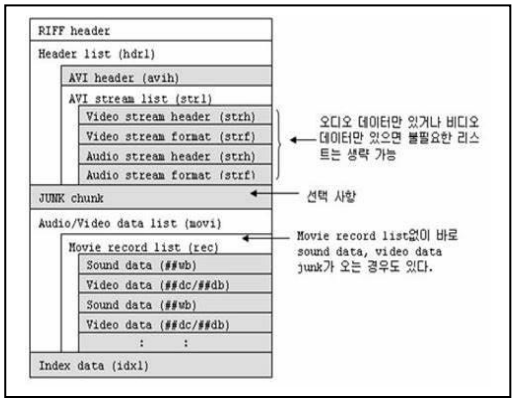
```
totalFrames = typecast(aviMainHeader(21:24), 'uint32'); % Total number of
frames
width = typecast(aviMainHeader(37:40), 'uint32'); % Width of the video
height = typecast(aviMainHeader(41:44), 'uint32'); % Height of the video

% Close the file
fclose(fileID);

% Display parsed header information
disp('AVI Header Information:');
fprintf('Chunk ID: %s\n', chunkID);
fprintf('Chunk Size: %d bytes\n', chunkSize);
fprintf('Format: %s\n', format);
fprintf('List ID: %s\n', listID);
fprintf('List Size: %d bytes\n', listSize);
fprintf('List Type: %s\n', listType);
fprintf('Microseconds Per Frame: %d μs\n', microSecPerFrame);
fprintf('Max Bytes Per Second: %d bytes/s\n', maxBytesPerSec);
fprintf('Total Frames: %d\n', totalFrames);
fprintf('Width: %d pixels\n', width);
fprintf('Height: %d pixels\n', height);
```

How It Works

1. **fopen**: Opens the AVI file in binary mode.
2. **fread**: Reads specific chunks of data.
3. **typecast**: Converts raw bytes into meaningful data types (e.g., uint32 for integers).
4. **Header Parsing**: Extracts key metadata from the AVI file.

The output will be :

AVI Header Information:
Chunk ID: RIFF
Chunk Size: 10485760 bytes
Format: AVI
List ID: LIST
List Size: 200 bytes
List Type: hdrl
Microseconds Per Frame: 33333 µs
Max Bytes Per Second: 1244160 bytes/s
Total Frames: 300
Width: 640 pixels
Height: 480 pixels



Additional Notes

- **Endianess**: AVI files are in **little-endian** format, which MATLAB handles automatically with typecast.
- **Extended Headers**: Some AVI files may have additional headers or extended formats.
- **Reading Frames**: For frame extraction, use the VideoReader class for simplicity.

Let me know if you'd like more advanced examples, such as extracting video frames or audio streams from an AVI file! 😊

Lecture 5 Graphics File Formats

MATLAB supports several graphics file formats for storing and exporting images. These formats include **BMP**, **JPEG**, **PNG**, **TIFF**, **GIF**, and others. MATLAB provides built-in functions to read, write, and manipulate these image file formats.

Animation Support:

- Use **GIF** format for animations.
- Example: Save a sequence of images as an animated GIF.

```
for k = 1:10
    frame = uint8(255 * rand(100, 100, 3)); % Random image
    [A, map] = rgb2ind(frame, 256);
    if k == 1
        imwrite(A, map, 'animated.gif', 'gif', 'LoopCount', Inf,
'DelayTime', 0.5);
    else
        imwrite(A, map, 'animated.gif', 'gif', 'WriteMode', 'append',
'DelayTime', 0.5);
    end
end
disp('Animated GIF saved as animated.gif');
```

Lecture 6 PDF File Format

The **PDF (Portable Document Format)** file format is a complex structure designed to represent documents in a platform-independent manner. Below is a high-level algorithmic explanation of how a PDF file is structured and how it can be parsed or generated. This algorithm is not tied to any specific programming language but provides a conceptual framework for understanding the PDF format.

Algorithm for Generating a PDF File

Input:

- A document with text, images, fonts, and other elements.

Output:

- A valid PDF file.

Steps:

1. **Initialize the PDF Header**:
    - Write the PDF version identifier (e.g., %PDF-1.7).
    - Add a comment line with non-ASCII characters (e.g., %âãÏÓ) to ensure the file is treated as binary.
2. **Create the Document Body**:
    - Define objects for each element in the document (e.g., pages, fonts, images).
    - Assign unique object numbers to each object (e.g., 1 0 obj).
    - Encode the content of each object using PDF syntax:
        - Text: Use streams and font definitions.
        - Images: Embed image data (e.g., JPEG, PNG) as streams.
        - Fonts: Include font definitions or references to standard fonts.
    - Organize objects hierarchically:
        - A **Catalog** object (root of the document).
        - A **Pages** object (list of pages).
        - Individual **Page** objects (content streams for each page).
3. **Build the Cross-Reference Table**:
    - Create a table that lists the byte offset of each object in the file.
    - Each entry in the table corresponds to an object number and its location in the file.

4. **Write the Trailer**:
    - Specify the location of the cross-reference table.
    - Include the root object (Catalog) and other metadata (e.g., document info, encryption details).
    - Add the %%EOF marker to indicate the end of the file.
5. **Optimize the File**:
    - Compress streams (e.g., using FlateDecode) to reduce file size.
    - Linearize the PDF (optional) for fast web viewing.

Algorithm for Parsing a PDF File

Input:

- A PDF file.

Output:

- Extracted content (text, images, metadata).

Steps:

1. **Read the PDF Header**:
    - Verify the PDF version identifier (e.g., %PDF-1.7).
    - Check for the binary comment line (e.g., %âãÏÓ).
2. **Locate the Trailer**:
    - Start from the end of the file and search backward for the %%EOF marker.
    - Extract the trailer dictionary to find the root object and cross-reference table.
3. **Parse the Cross-Reference Table**:
    - Read the byte offsets of all objects in the file.
    - Use the offsets to locate and load each object.
4. **Extract Document Content**:
    - Traverse the object hierarchy starting from the root object (Catalog).
    - Identify the **Pages** object and iterate through individual **Page** objects.
    - Decode content streams for each page to extract text, images, and other elements.

5. **Handle Compression and Encryption**:
    - Decompress streams (e.g., FlateDecode) if necessary.
    - Decrypt the file if it is password-protected (requires the correct password).
6. **Output the Extracted Content**:
    - Save text, images, and metadata in a usable format (e.g., plain text, JSON).

PDF File Header Format

```
%PDF-1.7
%âãÏÓ
1 0 obj
<< /Type /Catalog /Pages 2 0 R >>
endobj
2 0 obj
<< /Type /Pages /Kids [3 0 R] /Count 1 >>
endobj
3 0 obj
<< /Type /Page /Parent 2 0 R /Contents 4 0 R /Resources << /Font << /F1 5 0 R >> >> >>
endobj
4 0 obj
<< /Length 44 >>
stream
BT /F1 12 Tf 72 720 Td (Hello, World!) Tj ET
endstream
endobj
5 0 obj
<< /Type /Font /Subtype /Type1 /BaseFont /Helvetica >>
endobj
xref
0 6
0000000000 65535 f
0000000010 00000 n
0000000060 00000 n
0000000110 00000 n
0000000200 00000 n
0000000250 00000 n
trailer
<< /Size 6 /Root 1 0 R >>
startxref
350
```

```
%%EOF
```

Explanation of the Example

1. **Header**:
    - %PDF-1.7 specifies the PDF version.
    - %âãÏÓ ensures the file is treated as binary.
2. **Objects**:
    - Object 1: Catalog (root of the document).
    - Object 2: Pages (list of pages).
    - Object 3: Page (content and resources for a single page).
    - Object 4: Content stream (text "Hello, World!").
    - Object 5: Font definition (Helvetica).
3. **Cross-Reference Table**:
    - Lists the byte offsets of all objects.
4. **Trailer**:
    - Specifies the root object and the location of the cross-reference table.
5. **Content Stream**:
    - Displays "Hello, World!" at position (72, 720) using the Helvetica font.

Lecture 7 PPTX File Format

PPTX File format

The **PPTX file format** is the default file format for Microsoft PowerPoint presentations starting from PowerPoint 2007. It is part of the **Office Open XML (OOXML)** family of file formats, which are based on XML and ZIP compression. PPTX files are widely used for creating and sharing presentations that include slides, text, images, animations, and multimedia.

Key Features of PPTX

1. **XML-Based**:
   - PPTX files are composed of XML files stored in a ZIP archive. This makes them easier to parse and manipulate programmatically.
2. **Compression**:
   - The contents of a PPTX file are compressed using ZIP, reducing file size.
3. **Extensibility**:
   - The XML structure allows for easy extension and customization.
4. **Rich Media Support**:
   - PPTX supports embedded images, audio, video, and animations.
5. **Slide Layouts**:
   - PPTX files can include multiple slide layouts, themes, and master slides.
6. **Compatibility**:
   - PPTX files can be opened and edited by many software applications, including LibreOffice Impress, Google Slides, and Apple Keynote.

Structure of a PPTX File

A PPTX file is essentially a ZIP archive containing a collection of XML files and other resources. Here's the typical structure:

Key Files and Folders:

1. **[Content_Types].xml**:
   - Defines the types of content in the presentation (e.g., slides, themes, images).
2. **_rels/.rels**:
   - Specifies relationships between files in the package.
3. **ppt/presentation.xml**:
   - The main file that defines the presentation structure, including slide order and master slides.

4. **ppt/slides/slide1.xml**:
    - Contains the content of individual slides (e.g., text, shapes, images).
5. **ppt/slideLayouts/**:
    - Contains XML files defining slide layouts.
6. **ppt/slideMasters/**:
    - Contains XML files defining slide masters (templates for slides).
7. **ppt/theme/theme1.xml**:
    - Defines the theme (colors, fonts, and effects) used in the presentation.
8. **ppt/media/**:
    - Contains embedded media files (e.g., images, audio, video).
9. **ppt/notesSlides/**:
    - Contains notes associated with slides.
10. **ppt/comments/**:
    - Contains comments added to slides.

Example PPTX Workflow

Creating a PPTX File

1. **Create a Presentation**:
    - Use PowerPoint or another presentation tool to create slides with text, images, and animations.
2. **Save as PPTX**:
    - Save the presentation in PPTX format. This creates a ZIP archive containing XML files and resources.
3. **Inspect the PPTX File**:
    - Rename the .pptx file to .zip and extract its contents to view the XML files and folders.

Parsing a PPTX File Programmatically

Here's a high-level algorithm for extracting text from a PPTX file:

1. **Open the PPTX File**:
    - Treat the PPTX file as a ZIP archive.
2. **Extract the Contents**:
    - Unzip the archive to access the XML files and folders.
3. **Locate Slide Files**:
    - Navigate to the ppt/slides/ folder to find individual slide files (e.g., slide1.xml).
4. **Parse Slide XML**:
    - Use an XML parser to extract text and other elements from the slide files.

5. **Extract Media**:
    - Retrieve embedded media files from the ppt/media/ folder.
6. **Output the Content**:
    - Save the extracted text, images, and other content in a usable format.

Example: Extracting Text from a PPTX File (Python)

Here's a Python example using the python-pptx library to extract text from a PPTX file:

```
from pptx import Presentation

# Load the PPTX file
ppt = Presentation("example.pptx")

# Iterate through slides and extract text
for i, slide in enumerate(ppt.slides):
    print(f"Slide {i+1}:")
    for shape in slide.shapes:
        if hasattr(shape, "text"):
            print(shape.text)
```

Example: Creating a PPTX File (Python)

Here's how to create a PPTX file programmatically using python-pptx:

```
from pptx import Presentation

# Create a new presentation
ppt = Presentation()

# Add a slide with a title and content layout
slide_layout = ppt.slide_layouts[1]  # 1 is the layout index for "Title and Content"
slide = ppt.slides.add_slide(slide_layout)

# Add title and content
title = slide.shapes.title
title.text = "Hello, World!"

content = slide.shapes.placeholders[1]
content.text = "This is a sample PPTX file created using python-pptx."
```

```
# Save the presentation
ppt.save("example.pptx")
```

Tools for Working with PPTX

1. **Microsoft PowerPoint**:
   - The primary tool for creating and editing PPTX files.
2. **LibreOffice Impress**:
   - A free and open-source alternative to PowerPoint.
3. **Google Slides**:
   - A web-based tool for creating and editing presentations.
4. **Python Libraries**:
   - python-pptx: For creating and manipulating PPTX files programmatically.
   - unzip: For inspecting the contents of a PPTX file.
5. **Online Tools**:

Websites like Smallpdf and ILovePDF allow you to convert PPTX files to other formats (e.g., PDF).

Lecture 8 PNG File Format

PNG Image File Format

The **PNG (Portable Network Graphics)** file format is a widely used raster graphics format designed for lossless compression and efficient display of images on the web. It was developed as an improved, patent-free replacement for the GIF format. PNG supports a wide range of features, including transparency, gamma correction, and interlacing.

Key Features of PNG

1. **Lossless Compression**:
   - PNG uses the DEFLATE compression algorithm, which ensures no loss of image quality during compression.
2. **Transparency**:
   - PNG supports alpha channel transparency, allowing for smooth edges and translucent effects.
3. **Color Depth**:
   - PNG supports various color depths:
     - Grayscale (1, 2, 4, 8, 16 bits per pixel).
     - Truecolor (24-bit RGB, 48-bit RGB).
     - Indexed-color (1, 2, 4, 8 bits per pixel).
4. **Interlacing**:
   - PNG supports Adam7 interlacing, which allows images to display progressively in web browsers.
5. **Gamma Correction**:
   - PNG stores gamma information, ensuring consistent display across different devices.
6. **Metadata**:
   - PNG supports text chunks for storing metadata (e.g., author, description).
7. **Error Detection**:
   - PNG includes a CRC (Cyclic Redundancy Check) for detecting file corruption.

Structure of a PNG File

A PNG file consists of a series of **chunks**, each containing specific information about the image. The file begins with a **signature** and is followed by a sequence of chunks.

PNG File Signature

The first 8 bytes of a PNG file are always the same:

- Hexadecimal: 89 50 4E 47 0D 0A 1A 0A
- ASCII: ‰PNG␍␊␚␊

This signature identifies the file as a PNG image.

PNG Chunks

Each chunk has the following structure:

1. **Length** (4 bytes):
    - Specifies the length of the data field (not including the length, type, and CRC fields).
2. **Chunk Type** (4 bytes):
    - A 4-character ASCII string identifying the chunk type (e.g., IHDR, IDAT, IEND).
3. **Chunk Data** (variable length):
    - The actual data for the chunk.
4. **CRC** (4 bytes):
    - A checksum for the chunk type and data.

Critical Chunks

These chunks are required for a valid PNG file:

1. **IHDR (Image Header)**:
    - Contains basic information about the image:
        - Width, height, bit depth, color type, compression method, filter method, and interlace method.
2. **IDAT (Image Data)**:
    - Contains the compressed image data. There can be multiple IDAT chunks.

3. **IEND (Image End)**:
    - Marks the end of the PNG file.

Ancillary Chunks

These chunks are optional and provide additional information:

1. **tEXt (Textual Data)**:
    - Stores metadata as key-value pairs (e.g., author, description).
2. **iTXt (International Text)**:
    - Similar to tEXt but supports Unicode and compression.
3. **gAMA (Gamma)**:
    - Specifies the gamma value for the image.
4. **tRNS (Transparency)**:
    - Defines transparency for indexed-color or grayscale images.
5. **PLTE (Palette)**:
    - Contains the color palette for indexed-color images.
6. **bKGD (Background Color)**:
    - Specifies a default background color.
7. **pHYs (Physical Dimensions)**:
    - Specifies the intended pixel size or aspect ratio.

Example PNG File Structure

Here's a simplified example of a PNG file structure:

PNG Signature: 89 50 4E 47 0D 0A 1A 0A

IHDR Chunk:
- Length: 00 00 00 0D (13 bytes)
- Type: 49 48 44 52 ("IHDR")
- Data: Width (4 bytes), Height (4 bytes), Bit Depth (1 byte), Color Type (1 byte), Compression Method (1 byte), Filter Method (1 byte), Interlace Method (1 byte)

- CRC: 4 bytes

PLTE Chunk (optional):
- Length: 00 00 00 0C (12 bytes)
- Type: 50 4C 54 45 ("PLTE")
- Data: RGB palette entries (3 bytes per entry)
- CRC: 4 bytes

IDAT Chunk:
- Length: 00 00 00 0A (10 bytes)
- Type: 49 44 41 54 ("IDAT")
- Data: Compressed image data
- CRC: 4 bytes

IEND Chunk:
- Length: 00 00 00 00 (0 bytes)
- Type: 49 45 4E 44 ("IEND")
- Data: None
- CRC: 4 bytes

Tools for Working with PNG

1. Image Editors:
    - Adobe Photoshop, GIMP, Paint.NET.
2. Command-Line Tools:
    - pngcrush: Optimizes PNG files.
    - optipng: Reduces file size without losing quality.
    - pngcheck: Verifies the integrity of PNG files.
3. Programming Libraries:
    - Python: Pillow, png.
    - JavaScript: pngjs.
    - C/C++: libpng.
4. Online Tools:
    - TinyPNG: Compresses PNG files.
    - PNG to JPG/WebP converters.

Example: Reading a PNG File (Python)
Here's how to read and display a PNG file using the Pillow library in Python:

```
from PIL import Image

# Open a PNG file
image = Image.open("example.png")

# Display image information
print(f"Format: {image.format}")
print(f"Size: {image.size}")
print(f"Mode: {image.mode}")

# Show the image
image.show()
```

Example: Creating a PNG File (Python)
Here's how to create a PNG file programmatically using Pillow:

```
from PIL import Image, ImageDraw

# Create a blank image
image = Image.new("RGB", (200, 200), "white")
draw = ImageDraw.Draw(image)

# Draw a red rectangle
draw.rectangle([50, 50, 150, 150], fill="red")

# Save the image as a PNG file
image.save("example.png")
```

Lecture 9 GSM File Format

GSM File format

The **GSM file format** is primarily associated with **Global System for Mobile Communications (GSM) audio compression**. It is used to store audio data encoded using the GSM 06.10 codec, which is optimized for compressing voice data at a low bitrate. GSM files are commonly used in telecommunication systems, voicemail, and other applications where efficient voice data storage is required.

Key Features of GSM Files

1. **Compression**:
   - GSM 06.10 compresses audio to a bitrate of **13 kbps**, making it highly efficient for voice data.
2. **Sampling Rate**:
   - The standard sampling rate for GSM audio is **8 kHz**.
3. **Frame Structure**:
   - GSM audio is divided into frames, with each frame representing 20 ms of audio (160 samples).
4. **File Size**:
   - Due to its low bitrate, GSM files are much smaller than uncompressed audio formats like WAV.
5. **Compatibility**:
   - GSM files can be played back by many media players and converted to other formats.

Structure of a GSM File

A GSM file typically consists of a raw stream of GSM 06.10-encoded audio frames. There is no standard header or metadata, so GSM files are often considered "raw" audio files. However, some GSM files may include a **WAV header** to make them compatible with standard audio players.

Raw GSM File

- Contains only the GSM 06.10-encoded audio frames.
- Each frame is **33 bytes** long and represents **20 ms** of audio.

GSM in WAV Container

- A GSM file can be wrapped in a WAV container to include metadata and make it playable in standard media players.
- The WAV header specifies the audio format (GSM 06.10), sampling rate, and other details.

Example: GSM File Workflow

Creating a GSM File

1. **Record Audio**:
    - Record voice audio at 8 kHz sampling rate.
2. **Encode Audio**:
    - Use the GSM 06.10 codec to compress the audio into GSM format.
3. **Save as GSM**:
    - Save the encoded audio as a raw GSM file or wrap it in a WAV container.

Playing a GSM File

1. **Use a Compatible Media Player**:
    - Many media players (e.g., VLC) support GSM files directly.
2. **Convert to Another Format**:
    - Use tools like sox or ffmpeg to convert the GSM file to a more common format like WAV or MP3.

Tools for Working with GSM Files

1. **FFmpeg**:
    - A powerful command-line tool for converting and manipulating audio files.
    - Example: Convert a WAV file to GSM:

```
ffmpeg -i input.wav -ar 8000 -ac 1 -ab 13k output.gsm
```

SoX (Sound eXchange):

- A command-line utility for audio processing.
- Example: Convert a GSM file to WAV:

```
sox input.gsm output.wav
```

1. **Audacity**:
    - A free, open-source audio editor that supports GSM files with the FFmpeg plugin.

2. **VLC Media Player**:
    - A versatile media player that can play GSM files directly.

Example: Converting a WAV File to GSM (FFmpeg)
Here's how to convert a WAV file to GSM format using FFmpeg:

```
ffmpeg -i input.wav -ar 8000 -ac 1 -ab 13k output.gsm
```

Example: Creating a GSM File (Python)
Here's how to create a GSM file from a WAV file using pydub:

```
from pydub import AudioSegment

# Load the WAV file
audio = AudioSegment.from_file("input.wav", format="wav")

# Export to GSM
audio.export("output.gsm", format="gsm", bitrate="13k")
```

Lecture 10 WMV File Format

The WMV file format

The **WMV (Windows Media Video)** file format is a video compression format developed by Microsoft. It is part of the **Windows Media framework** and is widely used for streaming and downloading video content. WMV files are typically encapsulated in the **Advanced Systems Format (ASF)** container, which can also contain audio, metadata, and other multimedia elements.

Key Features of WMV

1. **Compression**:
   - WMV uses advanced compression algorithms to achieve high-quality video at relatively low bitrates.
2. **Streaming**:
   - WMV is optimized for streaming over the internet, making it suitable for online video delivery.
3. **DRM Support**:
   - WMV supports **Digital Rights Management (DRM)**, allowing content creators to protect their videos from unauthorized use.
4. **Compatibility**:
   - WMV files can be played on Windows Media Player and other media players that support the format.
5. **File Size**:
   - WMV files are smaller than uncompressed video formats like AVI, making them ideal for storage and distribution.
6. **Audio Support**:
   - WMV files often include audio encoded using the **Windows Media Audio (WMA)** codec.

Structure of a WMV File

WMV files are typically stored in the **ASF (Advanced Systems Format)** container. The ASF container is a flexible format that can store video, audio, metadata, and other multimedia elements.

ASF Container Structure

1. **Header Object**:
   - Contains metadata about the file, such as the title, author, and copyright information.

- Specifies the codecs used for video and audio.

2. **Data Object**:
   - Contains the actual video and audio data, organized into packets.
   - Each packet includes a timestamp for synchronization.
3. **Index Object** (optional):
   - Provides seek points for random access to the video.

WMV Video Codec

- The video stream in a WMV file is encoded using one of the WMV codecs:
  - **WMV 7**: Based on MPEG-4 Part 2.
  - **WMV 8**: Improved compression over WMV 7.
  - **WMV 9**: Introduced advanced features like interlaced video support and higher compression efficiency.
  - **WMV 9 Advanced Profile**: Supports high-definition video.
  - **WMV 9 Screen**: Optimized for screen recording.

WMA Audio Codec

- The audio stream in a WMV file is often encoded using the **Windows Media Audio (WMA)** codec, which provides high-quality audio at low bitrates.

Example: WMV File Workflow

Creating a WMV File

1. **Record or Import Video**:
   - Use a video editing tool or camera to capture video.
2. **Encode Video**:
   - Use a video encoder (e.g., Windows Media Encoder) to compress the video using a WMV codec.
3. **Add Audio**:
   - Encode the audio using the WMA codec and synchronize it with the video.
4. **Save as WMV**:
   - Save the video in the ASF container with a .wmv extension.

Playing a WMV File

1. **Use a Compatible Media Player**:
   - Windows Media Player, VLC, and other players that support WMV.
2. **Convert to Another Format**:
   - Use tools like FFmpeg or HandBrake to convert WMV files to other formats like MP4 or AVI.

Tools for Working with WMV Files

1. **Windows Media Player**:
    - The default media player for WMV files on Windows.
2. **VLC Media Player**:
    - A versatile media player that supports WMV files on multiple platforms.
3. **FFmpeg**:
    - A command-line tool for converting and manipulating video files.
    - Example: Convert a WMV file to MP4:

```
ffmpeg -i input.wmv output.mp4
```

4. **HandBrake**:
    - A graphical tool for converting video files, including WMV.
5. **Windows Media Encoder**:
    - A tool for encoding video and audio into WMV format.

Example: Converting a WMV File to MP4 (FFmpeg)

Here's how to convert a WMV file to MP4 format using FFmpeg:

```
ffmpeg -i input.wmv -c:v libx264 -c:a aac output.mp4
```

Example: Extracting Audio from a WMV File (FFmpeg)

Here's how to extract the audio from a WMV file and save it as an MP3:

```
ffmpeg -i input.wmv -q:a 0 -map a output.mp3
```

Example: Reading a WMV File (Python)

Here's how to extract metadata from a WMV file using the mutagen library in Python:

```
from mutagen.asf import ASF

# Load the WMV file
wmv_file = ASF("input.wmv")

# Print metadata
for key, value in wmv_file.items():
    print(f"{key}: {value}")
```

Example: Creating a WMV File (Python)
Here's how to create a WMV file from a video and audio file using moviepy:

```
from moviepy.editor import VideoFileClip, AudioFileClip, concatenate_videoclips

# Load video and audio
video = VideoFileClip("input_video.mp4")
audio = AudioFileClip("input_audio.mp3")

# Set audio to video
video = video.set_audio(audio)
# Export as WMV
video.write_videofile("output.wmv", codec="wmv2")
```

Lecture 11 SVG File Format

SVG File Format

The **SVG (Scalable Vector Graphics)** file format is a widely used XML-based vector image format for two-dimensional graphics. Unlike raster image formats (e.g., PNG, JPEG), SVG uses mathematical descriptions of shapes, lines, and colors, making it resolution-independent and ideal for scalable graphics like logos, icons, and illustrations.

Key Features of SVG

1. **Vector-Based**:
   - SVG images are composed of paths, shapes, and text, which are defined using mathematical equations. This allows them to scale infinitely without losing quality.
2. **XML-Based**:
   - SVG files are plain text files written in XML, making them human-readable and editable with a text editor.
3. **Interactivity**:
   - SVG supports interactivity through JavaScript and CSS, enabling animations, hover effects, and dynamic updates.
4. **Accessibility**:
   - SVG supports text descriptions and metadata, making it accessible to screen readers and search engines.
5. **Small File Size**:
   - SVG files are often smaller than raster images, especially for simple graphics.
6. **Wide Browser Support**:
   - SVG is supported by all modern web browsers.
7. **Animation**:
   - SVG supports animations using **SMIL (Synchronized Multimedia Integration Language)** or JavaScript.

Structure of an SVG File

An SVG file is an XML document that defines graphical elements using tags and attributes. Here's a basic example of an SVG file:

```
<svg width="100" height="100" xmlns="http://www.w3.org/2000/svg">
  <!-- Rectangle -->
  <rect x="10" y="10" width="80" height="80" fill="blue" />

  <!-- Circle -->
  <circle cx="50" cy="50" r="40" fill="red" />
```

```
  <!-- Text -->
  <text x="20" y="90" font-family="Arial" font-size="20" fill="white">SVG</text>
</svg>
```

Key Elements in SVG

1. **<svg>**:
    - The root element that defines the canvas and namespace.
    - Attributes:
        - width, height: Define the size of the canvas.
        - xmlns: Specifies the SVG namespace.
2. **<rect>**:
    - Defines a rectangle.
    - Attributes:
        - x, y: Position of the top-left corner.
        - width, height: Size of the rectangle.
        - fill: Fill color.
3. **<circle>**:
    - Defines a circle.
    - Attributes:
        - cx, cy: Center coordinates.
        - r: Radius.
        - fill: Fill color.
4. **<text>**:
    - Defines text.
    - Attributes:
        - x, y: Position of the text.
        - font-family, font-size: Font properties.
        - fill: Text color.
5. **<path>**:
    - Defines complex shapes using a series of commands (e.g., lines, curves).
    - Example:

```
<path d="M10 10 L90 90" stroke="black" stroke-width="2" />
```

6.<g>:

- Groups multiple elements together for easier manipulation.
- Example:

```
<g fill="green">
```

```
  <rect x="10" y="10" width="30" height="30" />
  <circle cx="50" cy="50" r="20" />
</g>
```

7. **<animate>**:
    - Adds animations to SVG elements.
    - Example:

```
<circle cx="50" cy="50" r="20" fill="orange">
  <animate attributeName="r" from="20" to="40" dur="2s" repeatCount="indefinite" />
</circle>
```

Example: Creating an SVG File

Here's how to create an SVG file manually:

1. Open a text editor (e.g., Notepad, VS Code).
2. Write the SVG code:

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
  <rect x="10" y="10" width="180" height="180" fill="yellow" stroke="black" stroke-width="5" />
  <circle cx="100" cy="100" r="80" fill="green" />
  <text x="50" y="120" font-family="Arial" font-size="40" fill="white">Hello</text>
</svg>
```

7.Save the file with a .svg extension (e.g., example.svg).

8.Open the file in a web browser or image viewer to see the result.

Tools for Working with SVG

1. **Vector Graphics Editors**:
    - Adobe Illustrator, Inkscape, CorelDRAW.
2. **Text Editors**:
    - VS Code, Sublime Text, Notepad++.
3. **Online Tools**:
    - SVGOMG: Optimizes SVG files.
    - Figma: A web-based design tool that supports SVG.
4. **Programming Libraries**:
    - JavaScript: D3.js, Snap.svg.

- Python: svgwrite, cairosvg.

Example: Manipulating SVG with JavaScript

Here's how to dynamically create and manipulate an SVG image using JavaScript:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SVG Example</title>
</head>
<body>
  <svg id="svgCanvas" width="200" height="200" xmlns="http://www.w3.org/2000/svg"></svg>

  <script>
    const svg = document.getElementById("svgCanvas");

    // Add a rectangle
    const rect = document.createElementNS("http://www.w3.org/2000/svg", "rect");
    rect.setAttribute("x", "10");
    rect.setAttribute("y", "10");
    rect.setAttribute("width", "180");
    rect.setAttribute("height", "180");
    rect.setAttribute("fill", "blue");
    svg.appendChild(rect);

    // Add a circle
    const circle = document.createElementNS("http://www.w3.org/2000/svg", "circle");
    circle.setAttribute("cx", "100");
    circle.setAttribute("cy", "100");
    circle.setAttribute("r", "80");
    circle.setAttribute("fill", "red");
    svg.appendChild(circle);
  </script>
</body>
```

```
</html>
```

Example: Creating an SVG File with Python

Here's how to create an SVG file programmatically using the svgwrite library in Python:

```
import svgwrite

# Create an SVG drawing
dwg = svgwrite.Drawing("example.svg", size=("200px", "200px"))

# Add a rectangle
dwg.add(dwg.rect(insert=(10, 10), size=(180, 180), fill="yellow", stroke="black", stroke_width=5))

# Add a circle
dwg.add(dwg.circle(center=(100, 100), r=80, fill="green"))

# Add text
dwg.add(dwg.text("Hello", insert=(50, 120), font_size=40, fill="white"))

# Save the SVG file
dwg.save()
```